UCRL-TR-218004

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Infiniband Performance Testing

M. Minich

January 6, 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Abstract

A look at the performance of the infiniband interconnect using the Voltaire host stack. This will attempt to compare not only infiniband to other high-performance interconnects, but will also take a look at comparing some of the different hardware choices available at the time of writing (e.g. Opteron, EM64T, pci-express and pci-x).

Contents

List of Figures

1 Overview and Goals

Infiniband promises to bring high performance interconnects for I/O (filesystem and networking) to a new cost performance level. Thus, LLNL will evaluate Infiniband as a possible cluster interconnect. Various issues impact the decision of which interconnect to use in a cluster. For this paper, we will stick to strictly looking at the performance of infiniband on different architectures. Performance testing will focus on latency, and bandwidth (both uni and bi-directional) using MPI. As a side result, manageability will largely be tested and we will strive to answer questions such as: "Was it difficult to configure and bring up a system with the tested infiniband interconnect?", "How difficult is it to upgrade the system and keep it running?".

1.1 Test Suite

All of the testing done in accordance with these results used the following benchmarks, which were made available from Ohio State University. More information for these tests (as well as the source code) can be found on the Ohio State Universities MPI over Infiniband Project website[1]. We are using these tests because they have become somewhat of a standard when used to benchmark mpi-based infiniband testing.

The compiling for each of these tests made use of a standard user environment with available tools. Running the resulting binary was performed through whatever standard mean was available for the system in question. More specfically, where available mpicc was used for compilation and srun or mpirun were used for runing the jobs. No additional flags were passed to the compilers, so the resulting binaries are non-optimized for the system they are running on. The reason for this was that we wanted to test what the generic results would be for an uneducated user who would simply want to use all of the default options to build an executable.

1.1.1 OSU Bandwidth Test

This test sends out a fixed number of back-to-back messages and waits for a reply from the receiver. When the receiver has collected all of the associated messages, the receiver sends a reply message back to the sender. Time is measured from the moment the sender sends it's first message to the time it receives a reply back. The test utilizes non-blocking MPI functions (MPI_Isend and MPI_Irecv).

1.1.2 OSU Latency Test

This test performs a ping-pong between a sender and receiver. A message of a given size is sent out, when a receiver receives the message it sends back a reply with the same data size. Averaging a number of iterations, one-way latency numbers are obtained. The test utilizes blocking MPI functions (MPI_Send and MPI_Recv).

1.1.3 OSU Bi-Directional Bandwidth Test

This test is similar to the bandwidth test except that both nodes send out back-to-back messages and wait for a reply. This is a measurement of the maximum sustainable aggregrate bandwidth by two nodes.

1.2 System Layout

Later we will go more in-depth in the actual software and hardware setup for the given test, but for now we can discuss the generic view of the basic cluster design that will be used throughout this paper. Figure 1 shows the basic layout of the cluster.

The overall design that we employ for our clusters is one with a large number of nodes who has the sole-purpose of computing in parallel utilizing a high-performance interconnect to provide

[1]See URL http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html

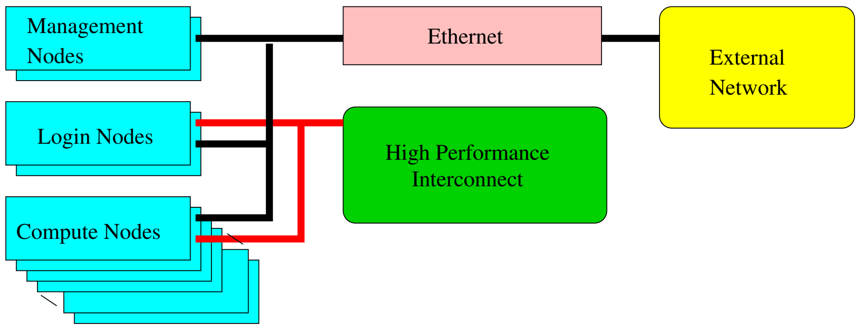


Figure 1: Generic System Layout

MPI layer communication. Each cluster has one or two management nodes that provide a centralized configuration point for the rest of the nodes. In addition, we use a number of dual-purpose login/routing nodes which provide a user interface into the cluster as well as a gateway for the compute nodes to speak with the rest of the computing center (e.g. to have access to the center-wide NFS home directories). At this point, each cluster has three internal networks. The first is the high-performance interconnect which is used for message-passing as well as providing a path for IO into and out of the cluster. The second is a management network (usually standard 100BaseT ethernet) which provides a backup lane for communicating to nodes as well as providing a path for installation and configuration of all of the nodes. The last network provides serial console and power-management access for total cluster management.

1.3 Operating System Software

Throughout this document you will probably see many references to ”chaos”, specifically pertaining to the operating system running on our clusters. Chaos is a derivative of the linux operating system produced by RedHat, and follows closely to what RedHat delivers. Chaos 1.2 was based on RH7.3 while chaos 2.0 is based on RHEL3 (RedHat Enterprise Linux 3). Chaos 3.0 (which is the latest version, but still in beta as of this writing) is based on RHEL4. Strictly speaking, chaos is a stripped down version of RedHat's operating system layered with extra cluster-related tools that are homegrown.

More information on Chaos can be found on the Linux@Livermore[2] website.

1.4 Price Comparison

A bit should be mentioned about the cost comparison of infiniband versus other technologies (especially Quadrics). The reason that this discussion is important is because it is a big part of the equation for deciding to switch from something like Quadrics to infiniband. In large cluster procurements, cost per node is a very important ratio as it determines how many nodes we can bring in (higher node counts with higher densities provide higher flop counts). In the linux world, the funds in a cluster procurement are generally first spent in determining the interconnect (based on what's available in the market and what can actually perform as expected), and the rest determines how many of each node type we can afford (along with the standard management hardware which falls into the noise).

Until recently, LC's production linux clusters primarily used the Quadrics (Elan3) interconnect. As an example, if we were to build a Quadrics cluster out of the smallest available switch, the switch would cost around $6,000 for a 16 port switch. If we were to upgrade this cluster to an Elan4 switch, the switch would then be $10,000 for a 16 port switch. Compare this to infiniband, with a 24 port switch costing $9,000. Looking at table 1, we can see the breakdown.

[2]See URL http://www.llnl.gov/linux

Table 1: Price Comparison of Elan3, Elan4, and Infiniband

	Elan3	Elan4	Infiniband
Switch	$6,000	$10,000	$9,000
# Nodes	16	16	24
Cost/Node	$375	$625	$375

You'll probably notice that with these figures, Elan3 and Infiniband are running at about the same price point, where as Elan4 just about doubles the price. Due to the amount of "in-the-field" time that Elan3 has, for Infiniband to be a cost-effective replacement, it is going to have to perform as well as Elan4 while staying at the Elan3 price-point.

2 Previous Testing

In October of 2004, a set of tests were performed on various systems in LC's I/O Testbed as well as a handful of other production resources. This previous testing is not exhaustive and does show some minor inconsistencies, however it is presented here to provide some general performance metrics of currently deployed technologies within LC. The software configuration of the systems in question isn't exact, but a good approximation.

2.1 Previous Test Hardware Configuration

A number of clusters were used for this initial testing. The basic hardware configuration can be seen in table 2. The most important pieces to notice are the differences in interconnects (Elan3, Elan4 and Infiniband). In addition, we will be able to see that there is a noticeable difference between the older Xeon-based systems and the newer.

Table 2: Previous Testing Hardware Configuration

System	Size	CPU	Memory	Interconnect
adev	16-node	dual 2.4GHz Xeon	4GB	Elan3
bigdev	128-node	dual 1.7GHz Xeon	2GB	Elan3
dev	16-node	dual 1.7GHz Xeon	2GB	Elan3
mcr	1152-node	dual 2.4GHz Xeon	2GB	Elan3
mdev	16-node	dual 2.4GHz Xeon (w/ HT)	2GB	Elan3
odev	11-node	dual 1.4GHz Opteron (32-bit)	4GB	Infiniband (PCI-X HCA)
tdev	8-node	quad 1.4GHz Itanium-2	8GB	Elan4
thunder	1024-node	quad 1.4GHz Itanium-2	8GB	Elan4

2.2 Previous Test Software Configuration

The basic software configuration for each of the test clusters can be seen in table 3. At the time of this testing, each of these systems were running the latest available chaos 2.0 release. The production machines were each at the latest stable kernel, while the testbed machines were at a higher revision. Odev was still behind at chaos 1.2 to allow for initial infiniband testing. Please note that chaos 2.0 is a modified version of RedHat Enterprise Linux version 3 while chaos 1.2 is a modified version of RedHat version 7.3.

Table 3: Previous Testing Software Configuration

System	Kernel	Interconnect Stack	MPI Version
adev	2.4.21-75chaos	qsnetlibs-1.4.3-1	qsnetmpi-1.24-8
bigdev	2.4.21-75chaos	qsnetlibs-1.4.3-1	qsnetmpi-1.24-8
dev	2.4.21-75chaos	qsnetlibs-1.4.3-1	qsnetmpi-1.24-8
mcr	2.4.21-71.3chaos	qsnetlibs-1.4.3-1	qsnetmpi-1.24-8
mdev	2.4.21-75chaos	qsnetlibs-1.4.3-1	qsnetmpi-1.24-8
odev	2.4.21-75chaos	ibhost-2.0.5_10	mvapich
tdev	2.4.21-75chaos	qsnet2libs-1.6.7-0	qsnetmpi-1.24-35.intel80
thunder	2.4.21-71.3chaos	qsnet2libs-1.6.7-0	qsnetmpi-1.24-35.intel80

3 Previous Testing Results

3.1 Bandwidth

Figure 2 shows the initial baseline results comparing bandwidth that is available on each of the systems tested. In this graph, the higher the datapoints, the better. First, we should notice the difference between Elan3, Elan4 and Infiniband (primarily, we'll use thunder, mcr and odev as reference points). At peak, mcr sees about half the bandwidth that is produced by thunder. This can be effectively understood when we remember that Elan3 has a theoretical peak of 350MB/s while Elan4 has a theoretical peak of 900MB/s (see Quadrics[3]). Next, we should pay attention to odev versus thunder. While still about 60MB/s slower than thunder, odev utilizes the much cheaper infiniband interconnect. The performance dip that occurs for odev at around the 1024B message size has yet to be explained. Further looking at the comparisons, we can see a major difference in peak between tdev and thunder (which should be the same). This could very well be a result of using a smaller 8-port Elan4 switch compared to thunders full-size switch. Also, you can see that there is a 100MB/s difference between the faster Xeon systems and the older systems. This is undoubtedly due to a faster front-side bus available on these systems.

3.2 Latency

Figure 3 shows the initial baseline results of latency results over each system tested by message sizes. In this graph, the lower the datapoints, the better. As we can see, Elan4 easily has the lowest latency over all of the interconnects tested. For larger message sizes, Infiniband begins to see a latency that is closer to Elan4. Otherwise, you can see that Elan3 is about 3x slower than Elan4 for the faster Xeon systems. The slower Xeon systems also see an even greater latency probably due to a slower front side bus. Again, it is important to notice that the Infiniband results are looking pretty good, especially when looking at price comparisons.

3.3 Bi-Directional Bandwidth

Figure 4 shows the initial baseline results of bi-directional bandwidth testing for each of the systems that we tested. Between Elan3 and Elan4, we can see about a 2x speed difference (again, the older Xeon systems see significantly lower speeds). Compared to the straight bandwidth test, we can see that the Elan3 and Elan4 systems show a slight increase in performance from this test. Additionally, we see that tdev is performing up around where it should be (compared to the results from the bandwidth test), but is still not quite up to par. It would seem that the 8-port Elan4 switch that is in use on tdev has some functional differences than the large Elan4 switches in thunder. Here we can see the first hints of Infiniband actually reaching a higher peak than Elan4. Apparently, this is only true for very large message sizes, but it should be interesting to see later comparisons.

[3]See URL http://www.quadrics.com

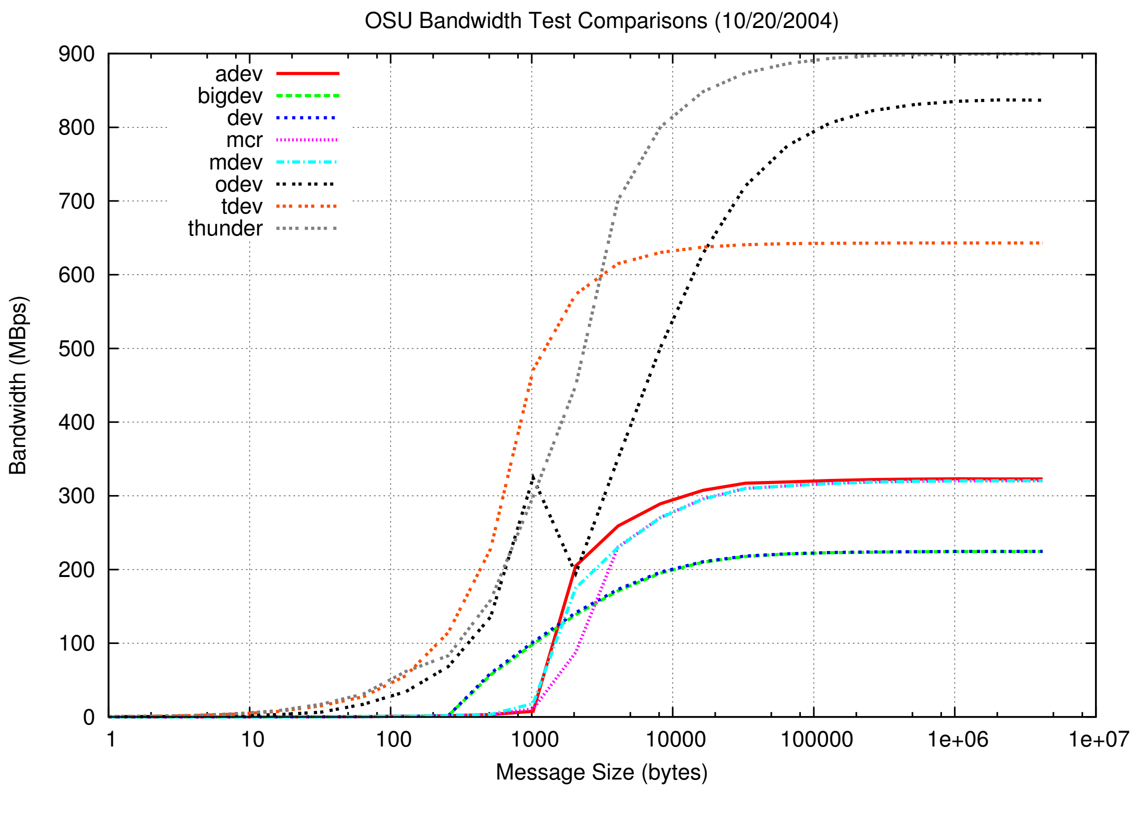


Figure 2: Previous Testing Bandwidth Results (higher is better)

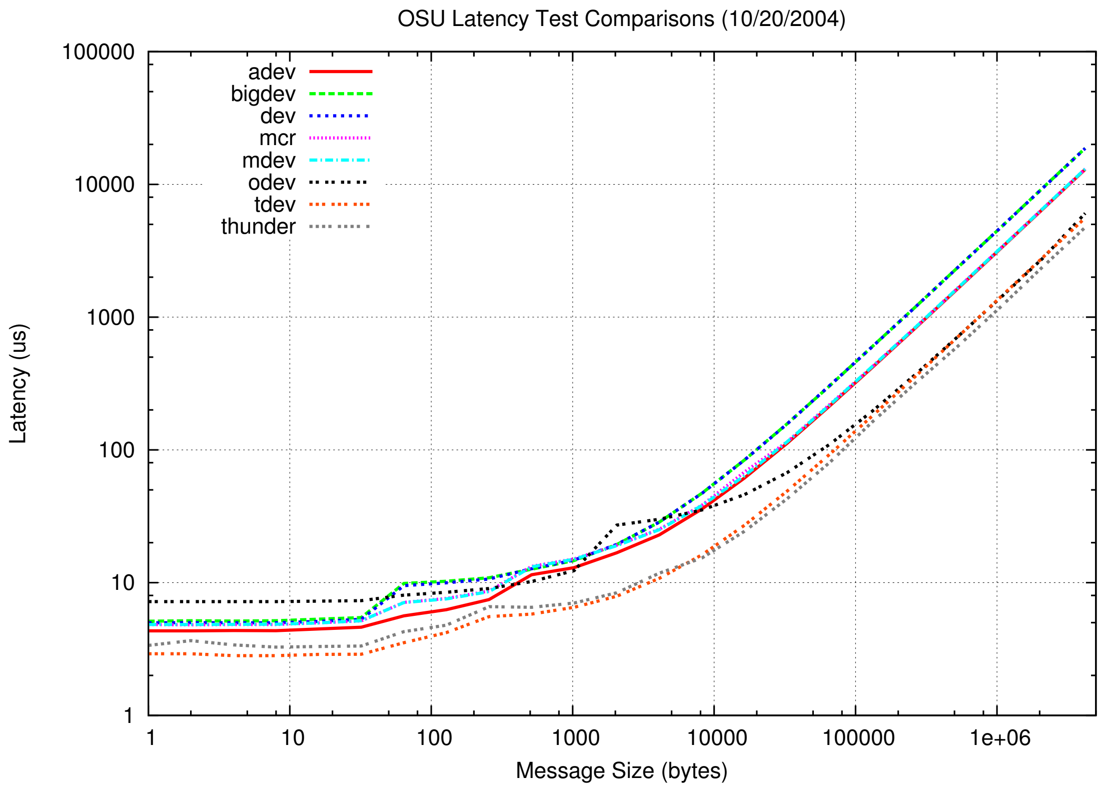


Figure 3: Previous Testing Latency Results (lower is better)

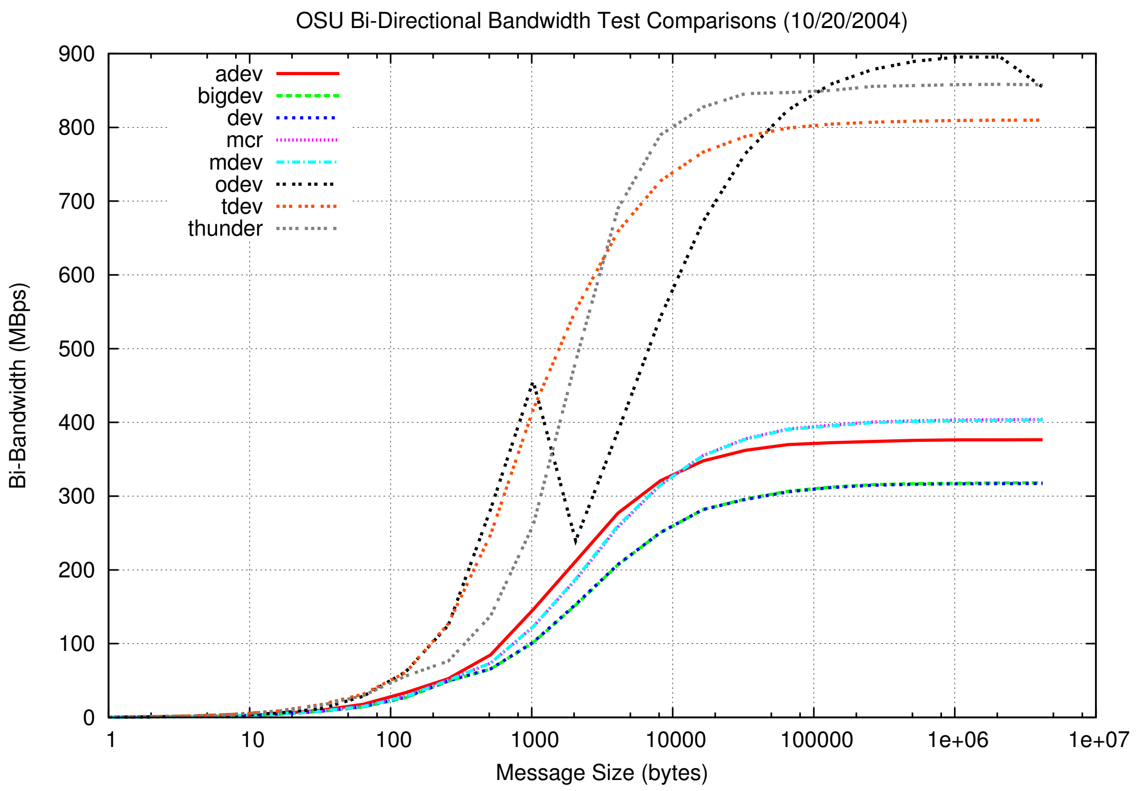


Figure 4: Previous Testing Bi-Direction Bandwidth Results (higher is better)

4 Infiniband Performance Testing

Now that we've looked at some previous testing, it is time to look more in-depth at infiniband configurations. Overall, we are looking to compare the effect that different hardware architectures can have on the ultimate performance of the infiniband interconnect.

4.1 Test Hardware Configuration

Tests will be conducted using 8 nodes–4 dual Intel EM64T nodes, and 4 dual AMD Opteron nodes–of ldev and 4 of the nodes previously used on odev. This should give us a good comparison of a number of hardware differences: pci-x vs. pci-express, x4 pci-express vs. x16 pci-express, Opteron vs. EM64T. Each node will be equiped with a Voltaire dual port infiniband adapter connected by a single port to a Voltaire infiniband switch.

Table 4: Test Hardware Configuration

System	Size	CPU	Memory	PCI Configuration
odev	11-node	dual 1.4GHz Opteron	4GB	PCI-X
ldev[0-3]	4-node	dual 3.4GHz EM64T w/HT	4GB	1-x4 PCI-e, 1-x16 PCI-e
ldev[4-7]	4-node	dual 2.2GHz Opteron	2.8GB	2-x16 PCI-e

All of these systems are capable of running in 32-bit or 64-bit mode, but all of the testing from here on out will be done in 64-bit mode. Also, due to the chipset on the 2.2GHz Opteron motherboard, only 2.8G of RAM is available, even though 4G is installed. This is due to the dual x16 pci-express slots reserving memory at the bios level (this can be somewhat analogous to the AGP slots of other motherboards reserving memory). As a frame of reference, odev is using a Voltaire ISR-9600 96-port switch while ldev is using the Voltaire ISR-6000 switch.

4.2 Test Software Configuration

All of the systems tested are running 64-bit installations of the Chaos linux operating system. At the time of testing, odev and ldev[4-7] were all running Chaos 3.0 (which is based on RedHat Enterprise Linux 4) while ldev[0-3] was running Chaos 2.0 (which is based on RedHat Enterprise Linux 3). Because of the change to Chaos 3.0, a vendor provided infiniband stack was not yet available, so we were forced to create a quick port of Voltaire's latest version of the IB stack. A comparison of the non-altered stack and the altered stack will be given in the next section. Table 5 shows the software levels that were tested.

Table 5: Test Software Configuration

System	OS	Kernel	Interconnect Stack	HCA Firmware
odev	Chaos 3.0	2.6.9-6.16.5llnlsmp	mellanox_tools-3.2.3 nvigor-3.0.0 mvapich-0.9.4	0x300000000
ldev[0-3]	Chaos 2.0-17	2.4.21-82chaos	ibhost-v3.0.0_16	0x400060002
ldev[4-7]	Chaos 3.0	2.6.9-6.16.5llnlsmp	mellanox_tools-3.2.3 nvigor-3.0.0 mvapich-0.9.4	0x400060002

While rebuilding the IB stacks for Chaos 3, it was determined that it is easier to manage the separated pieces that are part of the Voltaire stack (hence, there are three separate packages compared to the single ibhost package) rather than rebuilding the entire package. The mellanox_tools includes the mellanox drivers as well as the mellanox software tools, which include a firmware flash. The nvigor package contains the Voltaire specifics, which include the way to start and stop the connection (e.g. load/unload the modules, start ipoib). The last package, mvapich, contains a Voltaire-specific (not sure how specific, but a lot of the build scripts were modified by Voltaire) version of mpich version 1.2.6 with OSU's vapi additions (mvapich version 0.9.4). This includes mpi compilers as well as a way to run mpi programs.

4.3 Installation and Configuration of the IB Stack

This would be a good place to touch on what it takes to manage the infiniband stack on these specific clusters. Because we are attempting to go with as many default settings as possible, we will not go into a discussion about subnet managers, instead allowing the subnet manager that is built-in to the switch perform that duty (because of the size of clusters and testing we are talking about, the subnet manager's effects will be negligable anyway).

Now, you might be wondering why we seem to be focussing on Voltaire for these tests. The simple reason is to allow for a consistant testing environment while limiting the variables that can be introduced by other vendor host stacks. This paper is looking to focus on comparing infiniband to other interconnects and it is out of the scope of this paper to start comparing other infiniband host stacks. As an interesting point, it should be noted that at the very core, all current infiniband offerings are based on the Mellanox hardware and low-level Mellanox software drivers. For this reason, performance differences will be negligable at this level of testing (while management differences will be the greater focus).

Now, the greatest importance for this discussion is what it takes to go from an installed system to a system that can fully utilize the infiniband interconnect. Primarily all of this testing is based on a release of the Voltaire IBHOST stack. In this aspect, we should first mention a bit on how to build and install the stack (which will lead quite well into an explanation as to why a modified stack was needed in our environment). Focusing on ibhost-v3.0.0_16, here are the steps required to build a working rpm (starting with a source rpm):

1. Install the kernel source into /usr/src/<kernel_source>.
2. Compile the kernel source.
3. Create a link from /usr/src/<kernel_source> to /usr/src/linux.
4. Make sure that /etc/redhat-release (or equivalent) exists and is correctly matches your current system.
5. Build the entire ibhost stack.
6. Remove all of the files in /usr/mellanox, /usr/mst, /usr/voltaire.
7. Install the newly built ibhost rpm.

The biggest issue with all of this is that this must be built as root and will build in-place (e.g. the install process will clean out /usr/mellanox, install the mellanox source code into that directory and start building binaries). This causes a problem if you are attempting to build the ibhost package on a system that is currently using a previous version of ibhost since the building process will overwrite anything that you are currently using. Once all of the building is complete and the resulting packages are installed, we are back to using a normal RedHat system. In fact, the ipoib interfaces can be setup just like any other interface (e.g. the ipoib interface files are loacted in /etc/sysconfig/network-scripts/ifcfg-ipoib0) and the usage of the infiniband host stack is controlled through an initscript (chkconfig can turn voltaireibhost on and off).

Using this method, we need to create a new ibhost stack each and everytime a new kernel is released. While this isn't usually a problem in a stable environment, for new technologies this can be quite a hindrance as new kernels can be released quickly, requiring a rebuild of any needed kernel modules (which directly affects the infiniband stack). With Voltaire's packaging scheme, not only do we have to be careful which node we rebuild on (due to the overwriting of live and useable binaries), but it also requires the rebuilding of everything, not just the kernel modules, which is a timely process. To rebuild the mpi compilers each time a new kernel module is needed is a bit over-kill, which brings us to the reason for a modified stack.

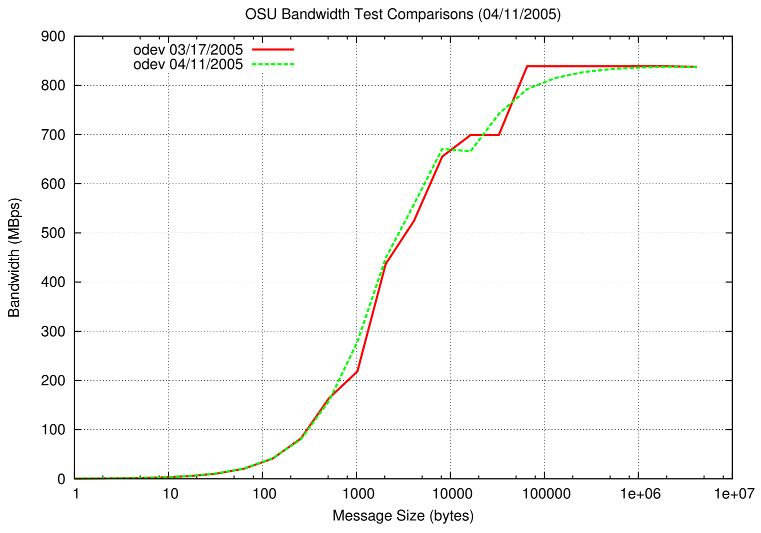
Not wanting to re-invent the wheel, but to create a more manageable build process, we found that there needed to be a new way to build the ibhost stack. This new stack does not change any fundamental work done in the ibhost stack, all it does is provide a safer way to build and install the same utilities. By design, the ibhost packages include the mellanox base tools (for HCA instantiation and firmware flashing), the Voltaire utilities, and a version of OSU's MVAPICH. It seemed a little more reasonable to go ahead and split these into their own packages. First, this provided a way to separate out the MPI implementation. This is helpful because it allows us to not have to rebuild MPI everytime we rebuild the kernel modules. In addition, we have the ability to change out the MPI version without touching the rest of the IB stack. Second, this new separation allowed for a way to redo the build process such that we no longer build in-place. By modifying the build process, any user can now build the entire stack without harming the running system.

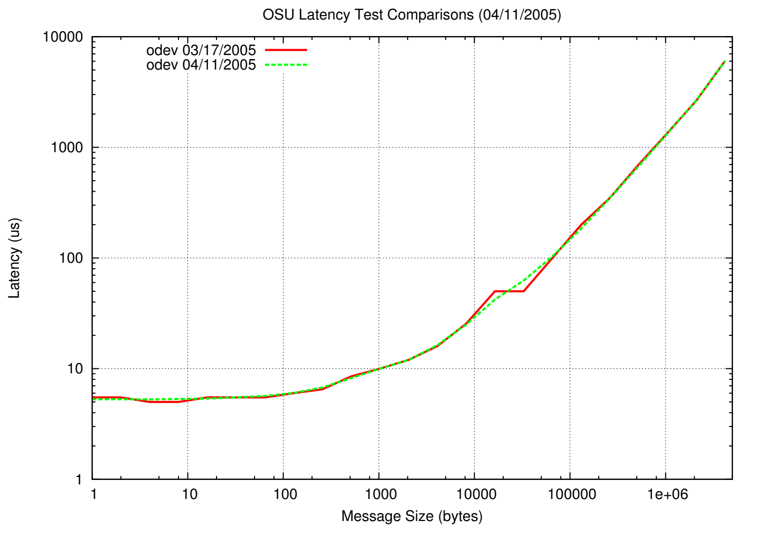
With either stack, configuration of the system is exactly the same. The Voltaire stack does a fair job of handling modules and options, provided you agree with them that the options that are being passed are correct. Rather than using the modprobe.conf file to pass options to needed modules, all module loading and application startup is handled by the startup scripts. While this seems a bit cleaner and better controlled, it mostly provides a difficult step when looking into debugging start-up issues. All configuration of the upper-level infiniband protocols (e.g. ipoib, srp, sdp) is handeled by configuration files in /etc/syscofnfig. If you don't care to use the standard RedHat paths for configuring your environment, you can use the helpful "ib-setup" tool to perform these configurations. All in all, getting past the building speed bumps, the Voltaire infiniband stack isn't very diffcult to configure and run (even after subsequent reboots of the system).

4.4 Comparing the Vendor Stack with the Modified Stack

Before we can start into more indepth comparisons of the current technologies, we first need to feel more assured that the modified stack can perform at least as well as the unmodified stack. In addition, it would be helpful to see whether the chaos upgrade from 2.0 to 3.0 has had any adverse affect on the infiniband performance. For this reason, we will first compare running with the previous stack to running with the modified stack. Using odev as the testcase, we will compare results performed before and after a software update. The results of this test can be seen in figure 5.

On 03/17/2005, odev was running chaos 2.0 with a 2.4.21-82chaos kernel and the ibhost-v3.0.0_16 Voltaire stack. On 04/11/2005, odev was running with its final software configuration (please see table 5 for more information). As you can see from the graphs, performance was pretty much unphased by the changes. If anything, the graphs became cleaner after the updates, with fewer sporatic jumps in performance. So, it seems safe enough to say that this new software will perform just as well as the previous.





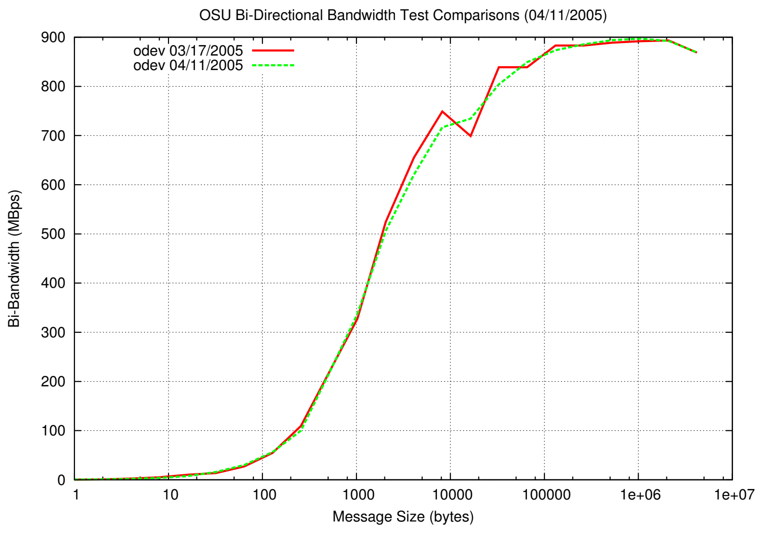


Figure 5: Stack Comparisons

5 Infiniband Performance Testing Results

In this first round of comparisons, we will take a look at the affect that the base system hardware can have on the infiniband performance. This section will specifically focus on comparing PCI-X vs. PCI-Express, x4 PCI-Express slots vs. x16 PCI-Express slots, as well as a comparison between Opteron and EM64T.

In each of these graphs, ldev-x16-ia32e refers to EM64T nodes with the HCA placed in a x16 PCI-Express slot, ldev-x4-ia32e referst to EM64T nodes with the HCA placed in a x4 PCI-Express slot and ldev-x86-64 refers to the Opteron based nodes. Please see table 4 for more information on the node types.

5.1 Bandwidth

In figure 6 we can see a visual comparison of the overall bandwidth between the different systems. The first thing to notice is the comparison between PCI-Express (both x4 and x16) and PCI-X. Looking specifically at peak, you can see that there is a noticeable impact that a x4 PCI-Express bus can create. Numerically, PCI-X performed at about 837MB/s, compared to x16's peak at about 970MB/s. On the otherhand, x4 PCI-Express peaked at about 748MB/s, about 90MB/s slower than PCI-X. One explanation for this is that a 4x Mellanox based infiniband card will only acheive full bandwidth by using at least an x8 PCI-Express slot that is physically wired to actually have x8 lanes.

Looking at the comparison between EM64T and Opteron, you can see that (ignoring the x4 results) both systems performed about the same. The EM64T saw a spike around 16K and 32K message sizes, but overall the two systems stayed pretty close.

5.2 Latency

The latency graph in figure 7 has been split into two pieces to allow a better view of the smaller packets (less that 0K) and the larger packets. It's easy to see that PCI-Express provides a lower latency than PCI-X. More interestingly, you can see that the x4 PCI-Express seems to still show some performance problems. In fact, on the average, the x4 PCI-Express sees about the same latency as PCI-X, while x16 PCI-Express sees a significant decrease in latency.

5.3 Bi-Directional Bandwidth

The last test is bi-directional bandwidth (see figure 8). This is where we can finally see PCI-Express shine. As expected, we can see that, even at x4, PCI-Express finally acheives a greater improvement in performance over PCI-X. In addition, we can see a slight lead given to the EM64T at higher packet sizes. Compared to the standard bandwidth, we're seeing about 2x speed up for bi-directional message passing.

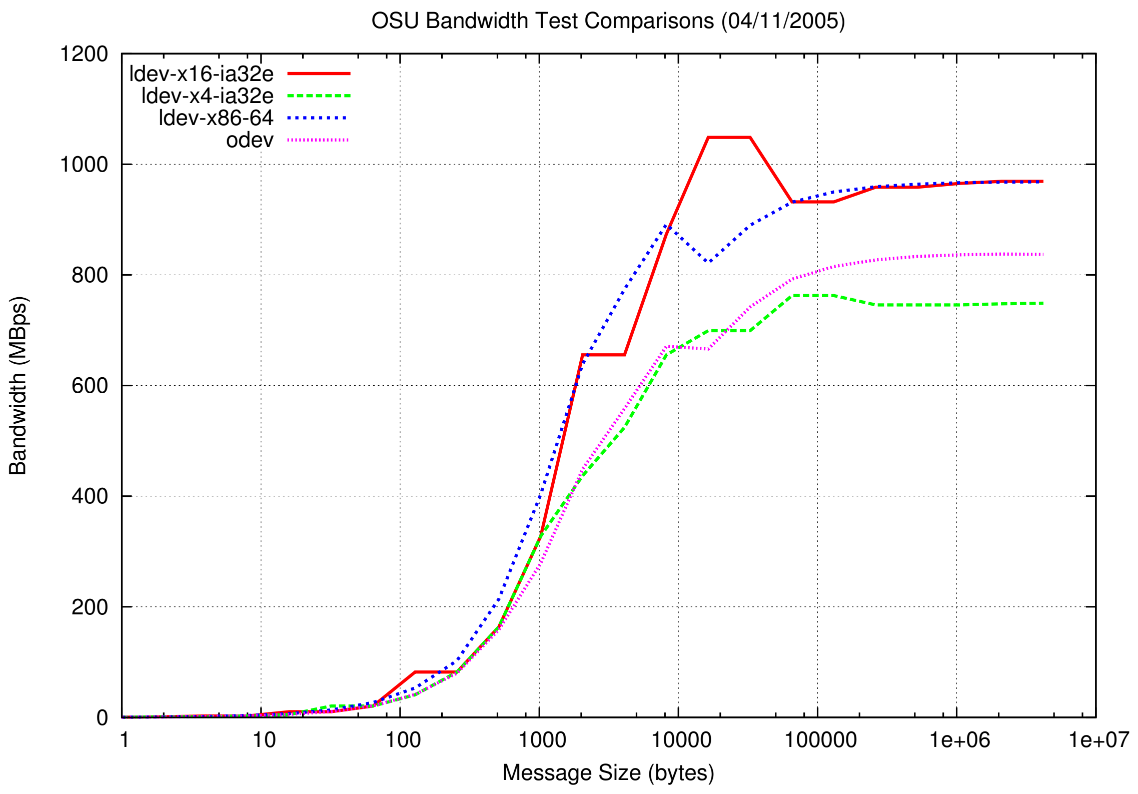
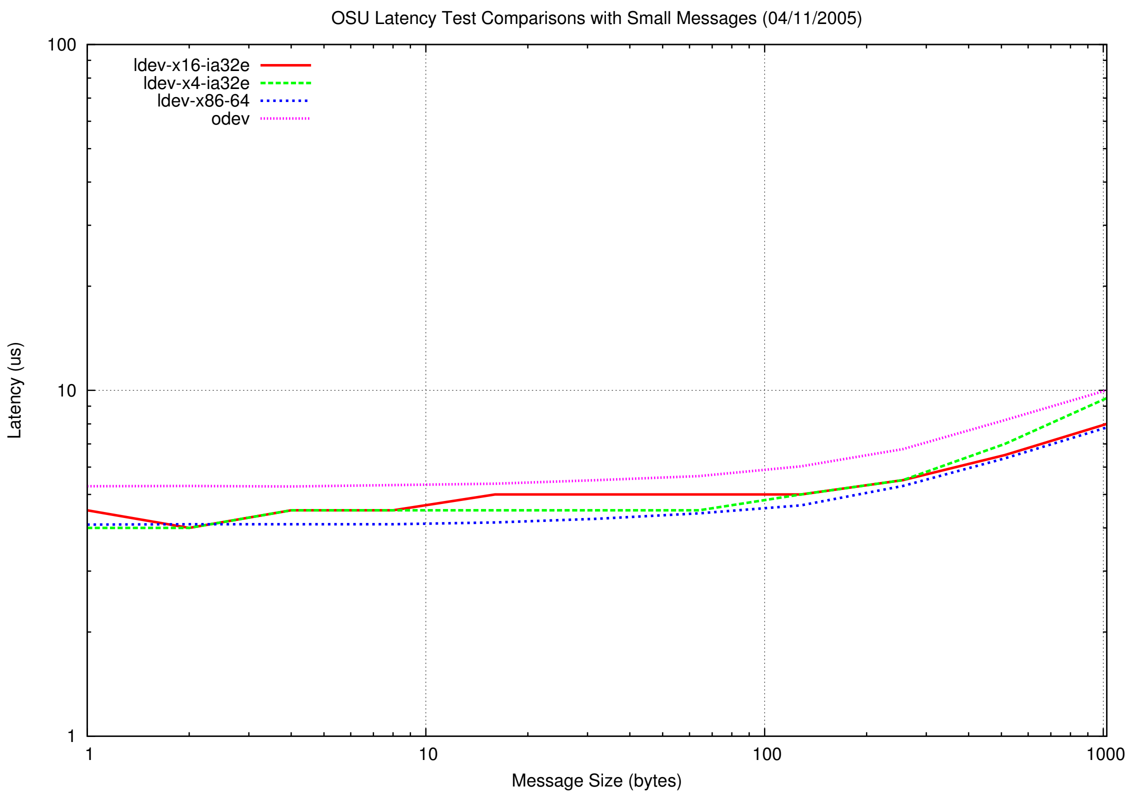


Figure 6: Infiniband Testing Bandwidth Results (higher is better)



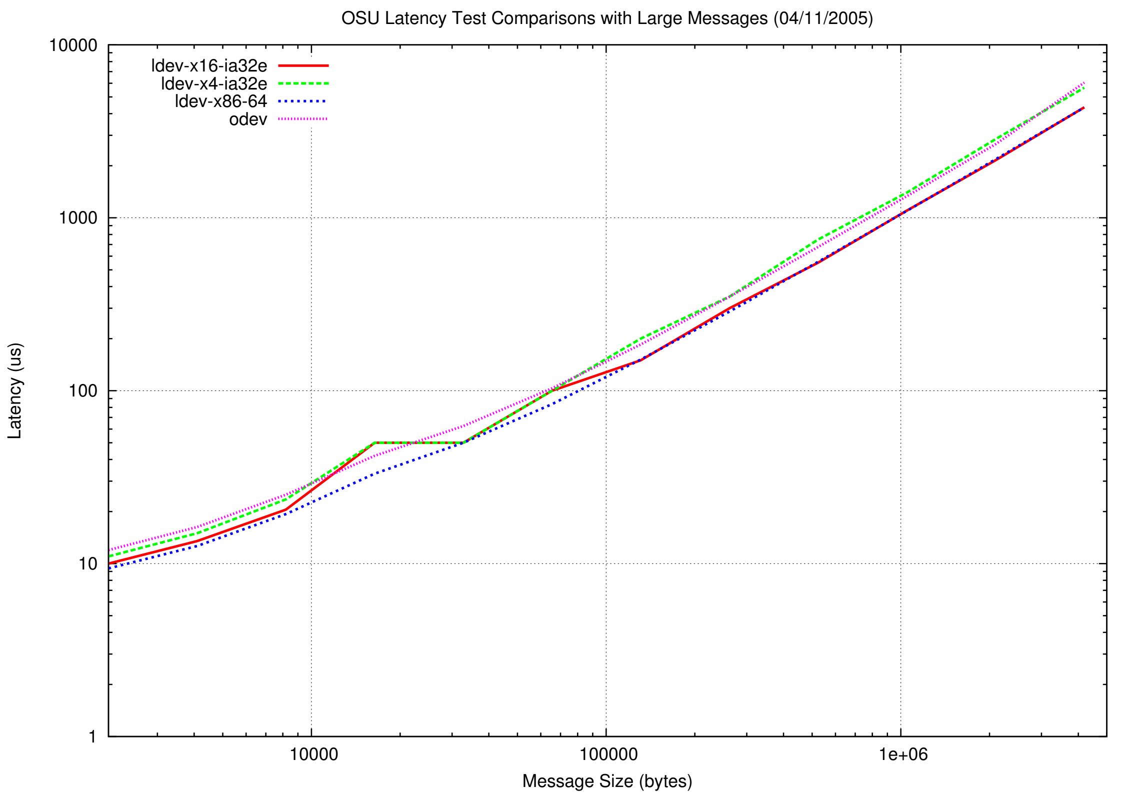


Figure 7: Infiniband Testing Latency Results (lower is better)

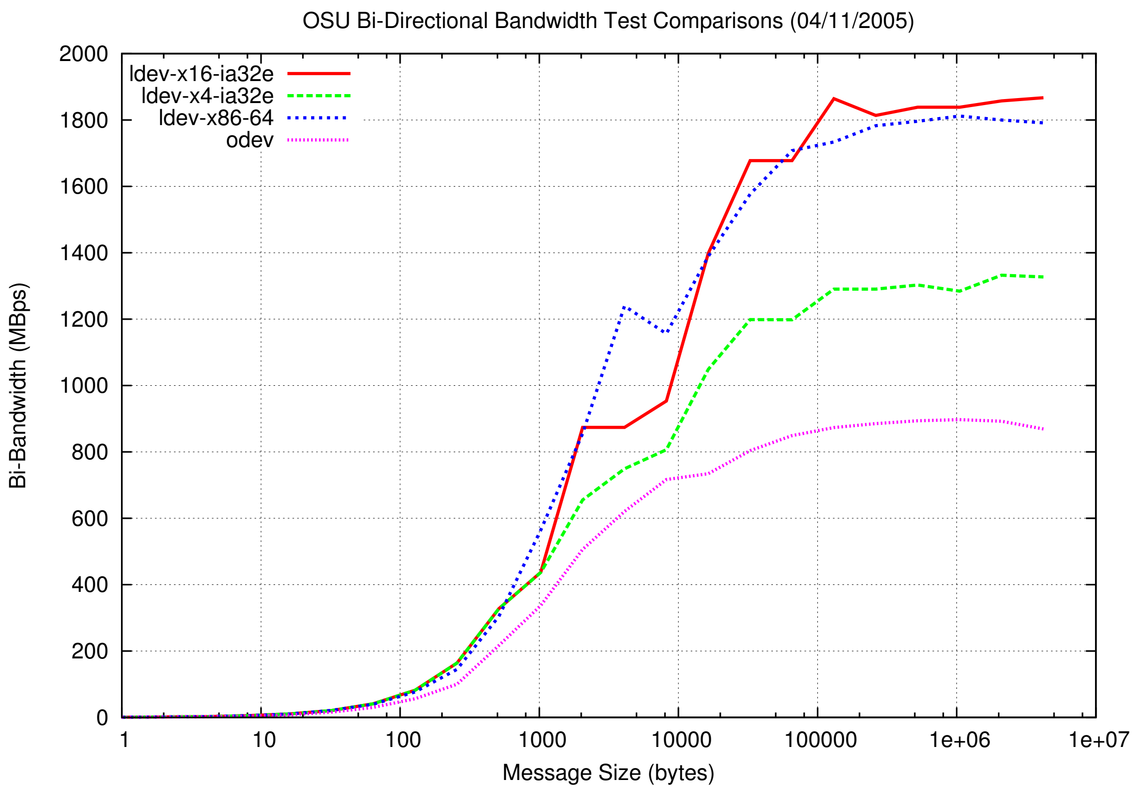


Figure 8: Infiniband Testing Bi-Directional Bandwidth Results (higher is better)

6 Final Comparison

So, lets take a look at a final comparison between all of the production systems and the updated infiniband clusters. This is where were going to see whether or not infiniband can really hold it's own in comparison to other high speed interconnects.

6.1 Bandwidth

From figure 9, it seems that with a peak of about 970MB/s compared to thunders 900MB/s, an Infiniband cluster utilizing x16 PCI-Express slightly outperforms Elan4. Though a 70MB/s speed difference isn't altogether that much, taking into account price comparisons this is a significant fact.

6.2 Latency

Comparing the latency of all of the systems, you can see that Elan4 still performs better than all of the other systems, specifically for smaller packet sizes (see figure 10). When the packet sizes get larger, the latency starts to even out. Again, taking into account price, it is pretty significant how close the latency on x16 PCI-Express is to Elan4.

6.3 Bi-Directional Bandwidth

The final comparison is with bi-directional bandwidth. From figure 11, we can see a significant increase with infiniband and PCI-Express. While elan4 sees a peak of about 860MB/s, x16 PCI-Express sees the peak at about 1.8GB/s. This is about a 2x speed difference, which is quite remarkable.

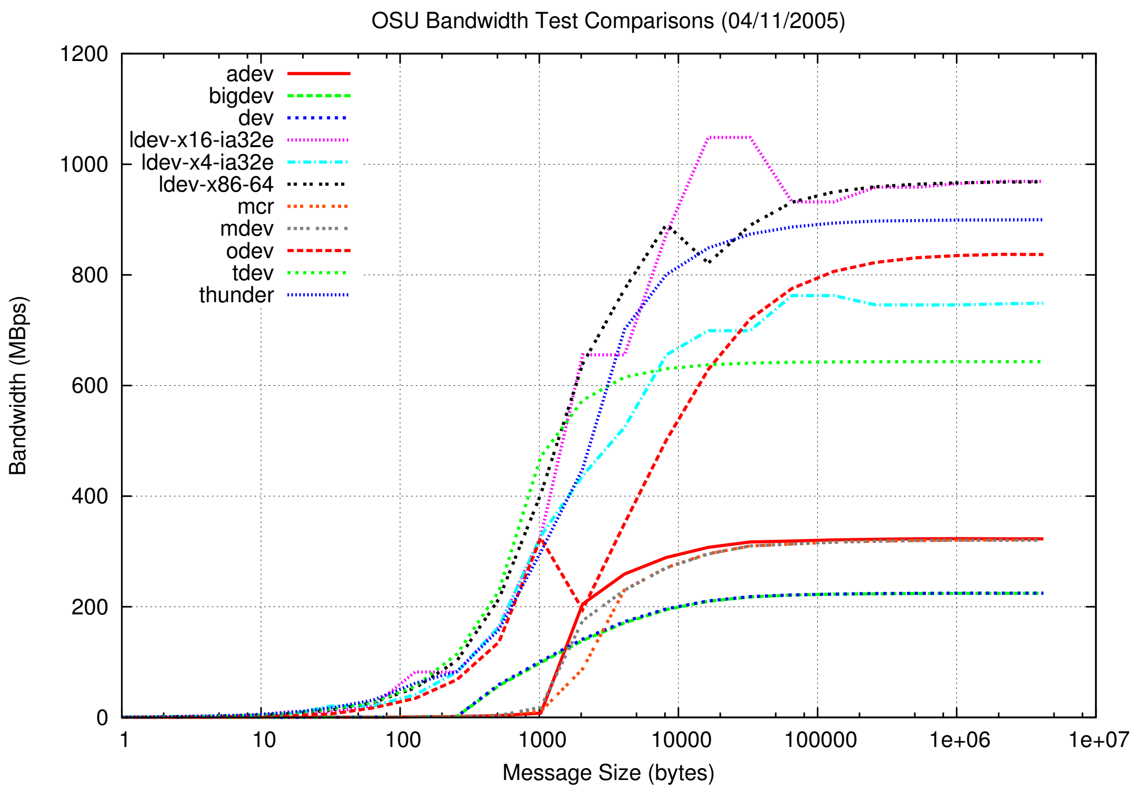
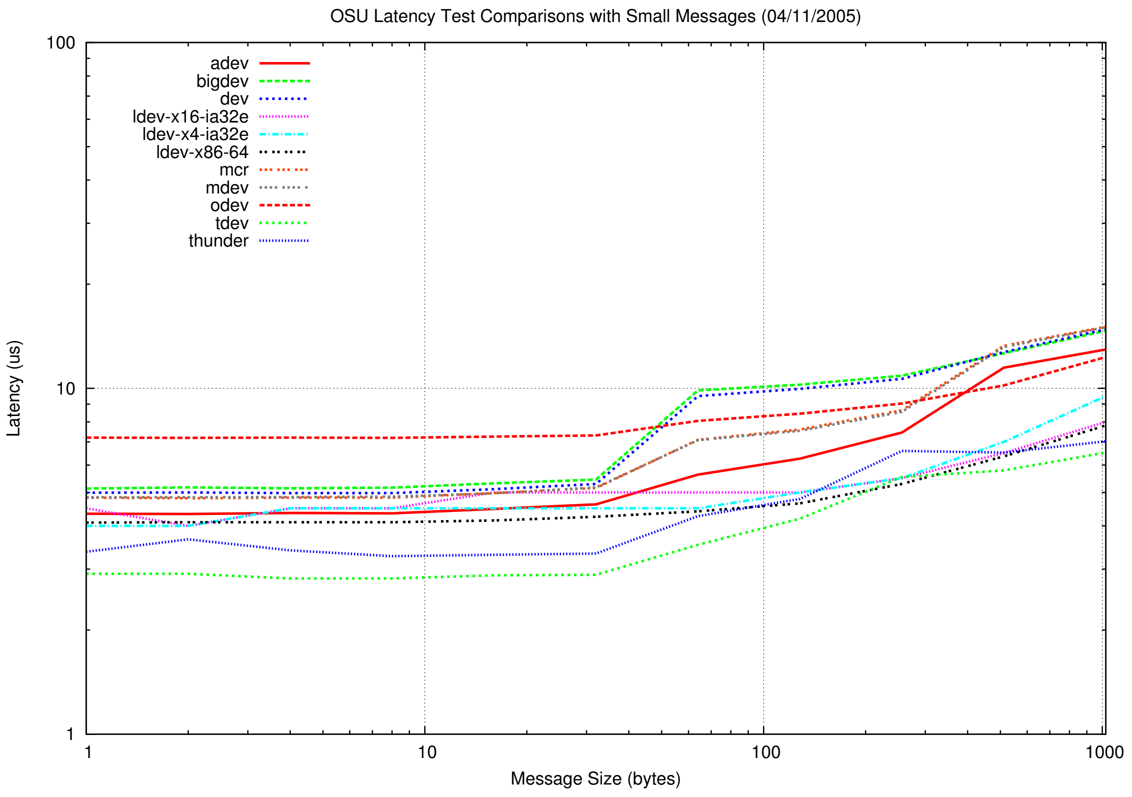


Figure 9: Comparison Bandwidth Results (higher is better)



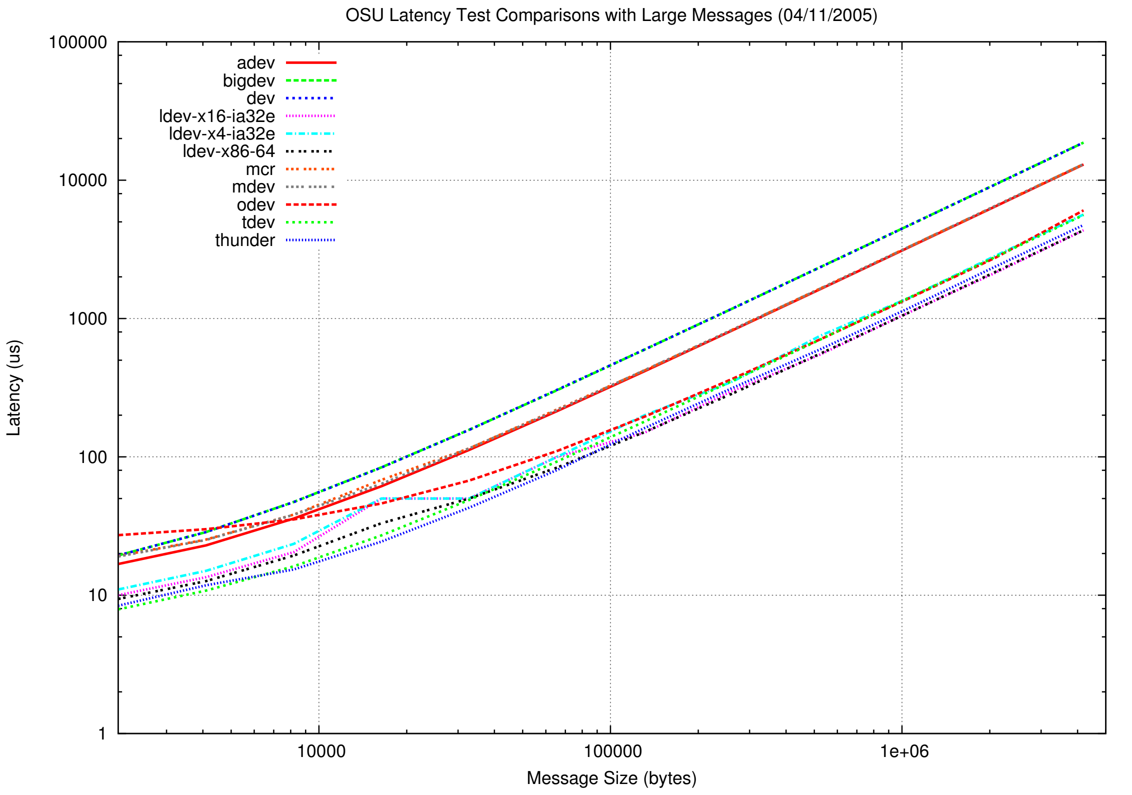


Figure 10: Comparison Latency Results (lower is better)

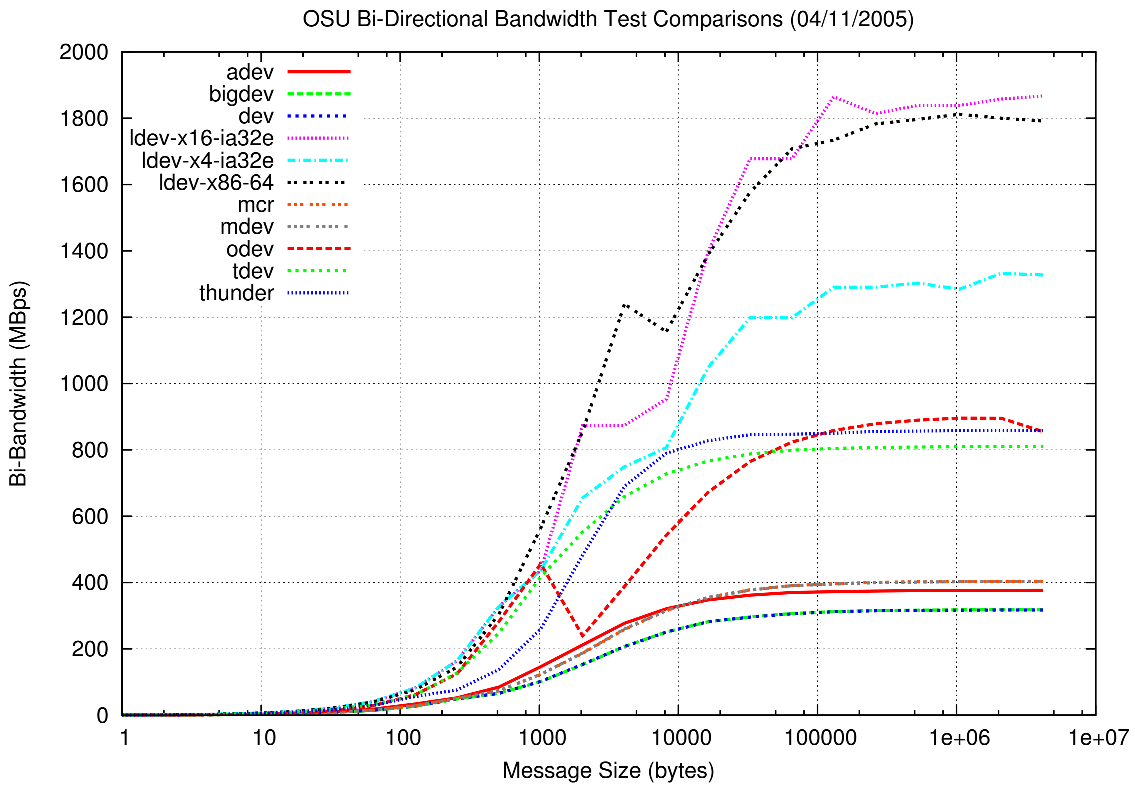


Figure 11: Comparison Bi-Directional Bandwidth Results (higher is better)

7 Conclusions

With comparable bandwidth and latency numbers and more than double the bi-directional bandwidth, all at a much lower per-node cost, it looks like infiniband is quite promising. As with any new technology, there are a number of kinks that need to be worked out, but it is quite an impressive initial showing.

From a systems administrator view, the Voltaire infiniband stack has a large number of kinks to work out (specifically in their build process) and could stand for a few tweaks in their underlying management system (relying more on existing linux standards like modprobe.conf and less on their own underlying scripts). Overall, once the initial hurdles are crossed an infiniband cluster is just as easy as any other cluster to install, configure and update packages. While there are many different underlying tweaks that one could attempt to make, relying on pure defaults allowed us to install, configure and run tests on an infiniband cluster quickly and reliably.